Developments in Strategies and Software Tools for Overset Structured Grid Generation and Connectivity

William M. Chan*

NASA Ames Research Center, M/S 258-2, Moffett Field, CA 94035

Recent developments in strategies and software tools for overset structured grid generation and domain connectivity are described. A scripting approach based on a library of grid generation script macros is adopted to speed up the grid generation process. Advances in this method involve the development of high level functions that encapsulate more complex grid generation tasks. Sample applications using the script library macros are given for several complex configurations. Advances towards automated and efficient domain connectivity is accomplished using a newly developed library of domain connectivity functions called Chimera Components Connectivity Library (C3LIB). The library contains standard interfaces to the various processes encountered in domain connectivity. Enhancement to the stencil search speed is accomplished via bounding boxes based on balanced index-space partitions, and replacement of global searches by local searches using space-filling curve segments. The performance of the parallel stencil search algorithm is illustrated via a set of test cases on practical complex configurations.

I. Introduction

Overset structured grid technology has been applied to increasingly complex aerospace applications in recent years.[1–6] The analysis procedure typically involves multiple steps and software tools. In addition to the flow solver used to compute the solution to the governing fluid flow equations, a number of pre-processing and post-processing tools are needed to perform the analysis. Typical pre-processing tasks include the creation of a CAD representation of the vehicle geometry, generation of surface and volume grids from the CAD model, performing domain connectivity between the overset grids, and preparing inputs to the flow solver. This input preparation step may include prescription of boundary conditions, numerical methods selection, as well as specification of prescribed or 6-dof motion dynamics. Typical post-processing steps include computation of aerodynamic integrated and line loads on various vehicle components, analysis of solution convergence, and visualization of the flow field.

Among all the pre-processing steps for structured overset grid computations, grid generation is by far the most time consuming. While the demand for grid generation on complex geometries has significantly increased, little effort has been invested by the community on automating the structured overset grid generation process for arbitrary geometries. The first part of this paper describes the strategy adopted by overset structured grid practitioners to handle the grid generation task under the current environment using a scripting approach. Recent advances in such an approach are presented.

The second part of this paper describes the attributes needed in a domain connectivity software to handle today's complex applications. This leads to the development of a library of domain connectivity functions with a standard interface. The current development is based on concepts for automating the object X-rays approach[7] but the library framework is built such that it can be easily extended to allow substitution of different algorithms for different steps of the domain connectivity process. Efforts to automate hole-cutting are described in a separate paper,[8] while efforts to improve the stencil search algorithm and various other steps are described in the current paper.

*Computer Scientist, AIAA Senior Member

II. Grid Generation Strategy and Tools

The most time consuming and labor intensive step in overset structured grid generation for complex configurations is in creating the surface grids. This task involves two main steps: the decomposition of the surface domain into overlapping quadrilateral patches that conform to the features of the surface geometry, and the distribution of grid points that would provide adequate resolution of both the geometry and the anticipated flow field. Once a high quality set of surface grids is created, generation of the body-conforming (near-body) volume grids is relatively simple using hyperbolic methods.[9]

Although the most difficult step in overset structured grid generation is at the beginning of the process in building the surface grids, there has been practically no research attempt to improve this task in the last ten or more years. The last known effort to the author's knowledge is described in Ref.10 where an algorithm was presented to automate surface domain decomposition around sharp and high curvature surface features. More recently, development of the Virtual Topology Editor (VTE) library,[11] and a CAD feature-tree-based scheme[12] are sparking new efforts to create general solutions to the surface domain decomposition problem. Further maturity of this technology is still needed before this is suitable for use in complex production cases.

While overset structured surface grid generation remains tedious and time consuming, the superior accuracy and speed of structured overset grid flow solvers[13] over those from other grid schemes make overset structured grids highly attractive to many applications practitioners. With no general automated scheme available to generate the surface grids, an alternate less general scheme has been adopted by various groups to speed up the grid generation process in many recent complex applications.[2–4, 6] This less general approach involves developing a script that contains all the steps needed to prepare all the inputs necessary for running the flow solver. These steps include the creation of surface grids from the CAD geometry, the generation of the near-body volume grids, the creation of various input files needed for domain connectivity, and the generation of input files for the flow solver and force/moment calculator.

The advantages of building a script that recreates the entire pre-processing procedure are many. Typical grid generation procedures for non-trivial geometries can contain tens to hundreds of steps. It would be extremely tedious for a user to have to go through a graphical user interface (GUI) to recreate the entire process for a small parameter change in one of the steps. In the scripting approach, various inputs to the grid generation process can be parameterized and the grid system can be automatically reproduced for different parameter settings. Typical parameters may include the physical dimensions, position, orientation of geometric components, the grid spacings at various locations in the domain, the maximum grid stretching ratio allowed, etc. Geometric components can be easily added or removed for different variations of a configuration. The final script also serves as a documentation for the various steps needed to create the grid system.

While the scripting approach permits rapid recreation of a process, a new script has to be manually created, or an existing script has to be significantly modified, for each new geometry encountered that is topologically different from previously scripted geometries. A topological change can be as small as an extra sharp-facet added to the geometry, to macroscopic modifications such as a wing/body junction where the wing changes from completely intersecting the body to only partially intersecting the body.

II.A. The Chimera Grid Tools Script Library

Several years ago, the creation of such pre-processing scripts was itself a non-trivial task since it involved numerous steps, with each step requiring the execution of a grid utility with multiple lines of input. This led to the development of a script library of functions that encapsulates each low level command with a one-line macro. Such a library was incorporated into the Chimera Grid Tools (CGT) software package. Chimera Grid Tools is a collection of software tools created specifically for efficient pre-processing and post-processing of structured overset grid computations. Pre-processing tools cover geometry and grid manipulation, grid generation, domain connectivity, and flow solver input preparation. Post-processing tools include force and moments computation,[14] line loads computation,[15] solution convergence analysis,[16] and solution visualization. A graphical user interface called OVERGRID[17] can be used to access most grid generation and manipulation tools. A library of evolving pre-processing macros (CGT Script Library) forms the basic building blocks for grid generation scripts.[18]

The CGT Script Library contains over 100 script macros based on the Tcl scripting language.[19] Each macro can be invoked by a one line call from a driver script. Inside each macro are wrappers that call one or more Fortran utility codes to efficiently perform the macro's function. Input and output grid files from all macros employ the PLOT3D format for structured grids, and the CART3D format for unstructured surface triangulations. The various pre-processing macros can be classified into four main categories: geometry creation, grid manipulation and generation, domain connectivity, and flow solver inputs preparation. These are described in more details in the following subsections.

II.A.1. Geometry Creation Macros

The geometry creation macros can be used to build up geometry quickly from a number of basic geometric entities. These macros allow creation of points, straight lines, parameterized analytic curves, rectangles, rectangular prisms, cylinders and frustums with truncated or rounded end caps. The intent here is not to duplicate the full geometric capabilities of commercial CAD software, but to allow a user to perform rapid experimentation on light-weight geometric configurations without having to go through the steep learning curve of most CAD packages. For some of these macros, the output grid file is not just the geometry definition but can be used directly as the computational grid. For example, the cylinders macro with the rounded end cap option has been utilized to rapidly create the strut tower grids for the Saturn-V launch escape system (Fig. 1).

Airfoil shapes are fundamental building blocks to aeronautical vehicles. In order to demonstrate the power and flexibility of the script library in rapidly creating various components of such vehicles, a specialized macro (*CreateNACAFoil*) has been developed to create a 2-D NACA 4-digit series airfoil[20] where the digits are specified by the user. By combining this macro with grid concatenation and redistribution macros, together with options for span, sweep angle, and dihedral angle specifications, a 3-D airfoil component macro (*CreateAirfoilComponent*) has been built to create structured surface geometry/grid files for basic airplane components such as wings, tails, fins, etc. Also, a combination of *CreateNACAFoil* with the grid revolution macro has been used to generate a nacelle. With under ten macro calls, a complete subsonic airplane geometry could be generated using combinations of the frustum macro with rounded ends for the fuselage, and the *CreateAirfoilComponent* macro for the wings, nacelles, horizontal and vertical tails (Fig. 2a). Figs. 2b and 2c show various other combinations of such macros to create a supersonic airplane, and a fictional X-wing vehicle, respectively.

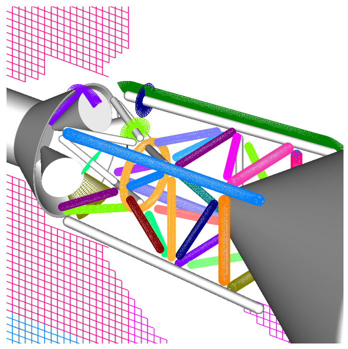


Figure 1. Saturn-V launch escape system strut tower grids created by the cylinder macro.

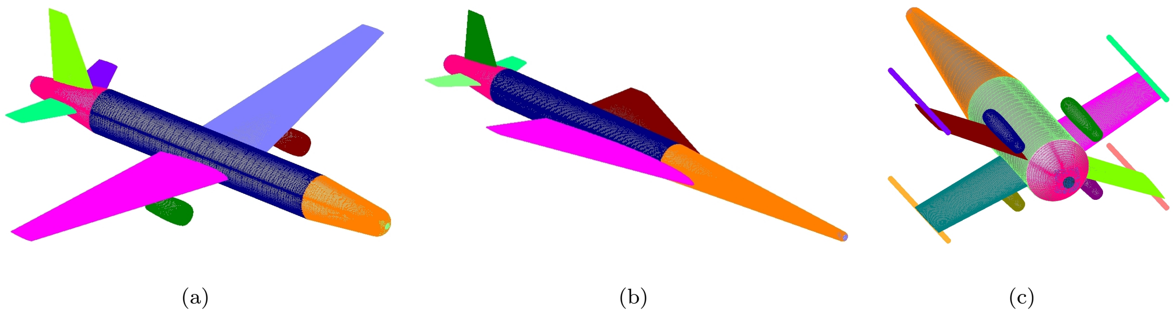


Figure 2. Various aerospace vehicle geometries created by combinations of geometry creation script macros (a) Subsonic airplane. (b) Supersonic airplane. (c) Fictional X-wing vehicle.

II.A.2. Grid Manipulation and Generation Macros

The majority of the macros in the CGT Script Library have been developed for grid manipulation and generation. These macros can be classified into four categories of increasing complexity and sophistication:

1. *Level 1 macros* - Early in the development of this library, the concern was with low level operations needed for grid generation. Such macros include grid subset extraction, concatenation, index reversal and swapping, grid translation, rotation, scaling, and mirroring, grid point redistribution, projection, and smoothing, hyperbolic surface and volume grid generation, algebraic surface grid generation, Cartesian volume grid generation, and others.
2. *Level 2 macros* - As the grid generation tasks became more complex, a Level 2 set of macros were constructed to perform more complex functions that involve specific combinations of Level 1 macros. Examples of Level 2 macros include creation of a cap grid over a singular axis point on a surface grid (*CreateAxisCap*), creation of surface and volume grids for an axisymmetric body from a prescribed symmetry plane curve (*BuildAxisymGrids*), and creation of a plume grid from a given arbitrarily oriented nozzle grid (*BuildGeneralPlumeGrids*).[18]
3. *Level 3 macros* - More sophisticated macros that include expert knowledge in grid generation decisions are labeled as Level 3 macros. Currently, only one such macro has been constructed in the CGT Script Library for automatic redistribution of grid points on a multi-segment curve.[18] Grid points are automatically distributed at interior convex and concave corners and high curvature regions based on simple user inputs that specify maximum grid stretching, maximum turning angle, and global and end grid spacings.
4. *Level 4 macros* - At an even higher level still, Level 4 macros can be developed that create grid systems for specific classes of geometries such as wings, capsules, rotor blades, etc. The earliest attempt at creating a Level 4 macro was the collar grid script used for making collar grids for resolving wing/body junctions on airplane-like geometries.[21] Although it was successful at producing satisfactory collar grids at the time, it soon became clear that it is a much more difficult task to write a general collar grid generation macro for arbitrary intersecting components. For example, the original collar grid script would fail if the wing was a high or low wing that only partially intersected the body. Also the script would fail if there are sharp surface features on one or both of the intersecting components that need to be followed by the collar grid.

More recently, users of the CGT Script Library have developed a Level 4 macro to study the effects of grid resolution on space capsules simulations[22] (Fig. 3a). Starting from a symmetry plane curve as the geometry definition, a grid system consisting of a main axisymmetric body with two end caps, and an off-body Cartesian grid is created. In rotorcraft applications, Level 4 macros have been built for an N-bladed (N=3 or 4) rotor-hub system[23] (Fig. 3b). Each blade contains a main body O-grid with cap grids at the root and tip.

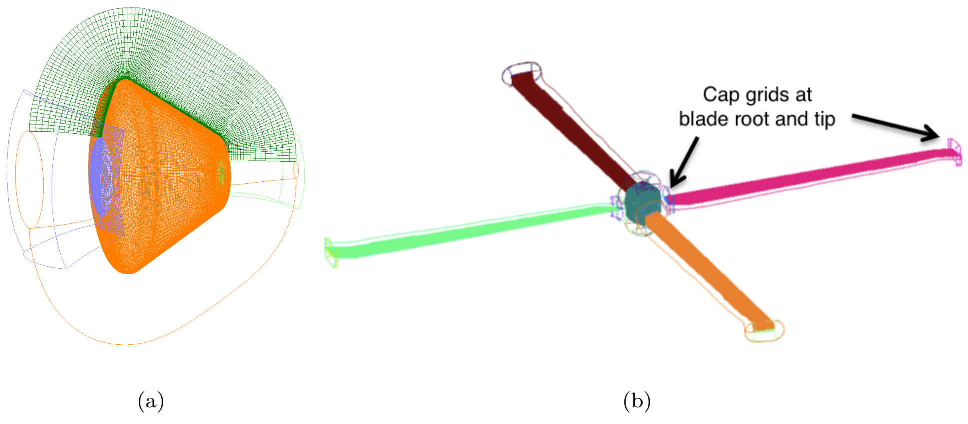


Figure 3. Examples of grid systems created using Level 4 script macros (a) Space capsule. (b) Rotor blades and hub.

II.A.3. Domain Connectivity Macros

A number of macros have been developed in the CGT Script Library to handle domain connectivity functions using object X-rays.[7] Object X-rays contain a rectangular array of ray pierce points on the surface of geometric components that are used in performing efficient hole-cutting. The macros available include the creation of an X-ray map from a given set of surfaces that describe a component (*CreateXrayMap*), the combination and various manipulations of X-rays (e.g., *CombineXrays*), and the specification of X-ray hole cut instructions (*SetOvrHoleCutInstr*). This last macro is used to prescribe the list of grids to be cut by each X-ray. Information for each instruction is stored in a global script array which is utilized to generate the X-ray domain connectivity inputs currently used inside the OVERFLOW flow solver.[13]

One of the most tedious tasks in the object X-rays approach to hole-cutting is in specifying the list of grids to be cut by each X-ray. This step becomes very difficult to manage when the number of X-rays and grids grow to hundreds as in today's complex applications. If new grids are added, or if existing grids are deleted or rearranged, all grid numbers that go into the X-ray hole cut specifications have to be modified, making this an extremely tedious and error-prone operation for the user. Before an automated method is available as proposed in Ref. 8, a scripting approach has been developed to make this step more manageable. This is achieved by associating names to groups of grids that are to be cut by the X-rays, as well as to the X-rays themselves. The grid numbers assoicated to each grid group are automatically recorded in a script array as all the grids in the system are appended into a multiple grid file. The grid numbers needed in the X-ray hole cut specifications are then automatically updated. This practice has been found to be highly worthwhile in building the domain connectivity inputs for complex grid systems (see examples in Section II.B).

II.A.4. Flow Solver Inputs Macros

Two macros are available in the CGT Script Library to handle flow solver inputs. The first is the *SetOvr-BCInput* macro that specifies the I^{th} boundary condition of the N^{th} grid. This information is stored in a global script array which is used by the *WriteOvr2Inputs* macro that writes an input file for the OVERFLOW flow solver. This second macro also takes the information from the global script array generated by *SetOvrHoleCutInstr* and writes the X-ray hole cut instructions to the OVERFLOW input parameters file.

II.B. Recent Applications

The CGT script library has been successfully applied to various complex configurations in the past several years. Some of these are presented in this section. In all cases, the starting point was a solid model geometry description in native CAD format. The *srf2cad* tool in CGT, linked to the CAPRI library,[24] was used to generate an intermediate high fidelity reference geometry file in an unstructured surface triangulation format and also output all CAD edges in a structured grid format. Using the surface triangulation and CAD edges as input geometry, a script was developed to create all surface and volume grids, the X-ray maps of various components for hole-cutting, the X-ray hole-cut instructions to indicate the grids cut by each X-ray, and the flow solver boundary conditions for the OVERFLOW code. Grid attributes such as maximum stretching ratio, and grid spacings at key locations are parameterized in the script for rapid regeneration of the entire process if needed. For the four complex configurations presented in the subsections below, the typical script development time was between two to four weeks.

II.B.1. Crew Launch Vehicle Integrated Stack

Grid generation scripts based in the CGT Script Library were heavily used by the NASA Ames Ares-I high-fidelity simulations team to model several generations and modifications of designs of the Ares-I crew launch vehicle. Fig. 4 shows one such design that includes the axisymmetric baseline geometry for the launch abort vehicle, the upper-stage, inter-stage and first-stage (Solid Rocket Booster), together with various protuberances such as the launch abort nozzles, different feedlines, ullage, tumble and booster-deceleration motors, various reaction control systems, antennas, cameras, and hold-down structures for the first-stage. Component scripts were built for each protuberance that can be executed independently from each other. A master script was then developed to assemble all the different protuberance grids and X-rays, and generate all necessary input files for the flow solver. The final grid system typically consisted of about 190 viscous volume grids, 185 million grid points for flight Reynolds number conditions, and 89 component X-rays.

II.B.2. Crew Launch Vehicle in Stage-Separation Mode

For the Ares-I crew launch vehicle during separation of the first stage from the vehicle, extensions of the component scripts were developed to model the exposed base region and nozzle of the upper-stage, and the

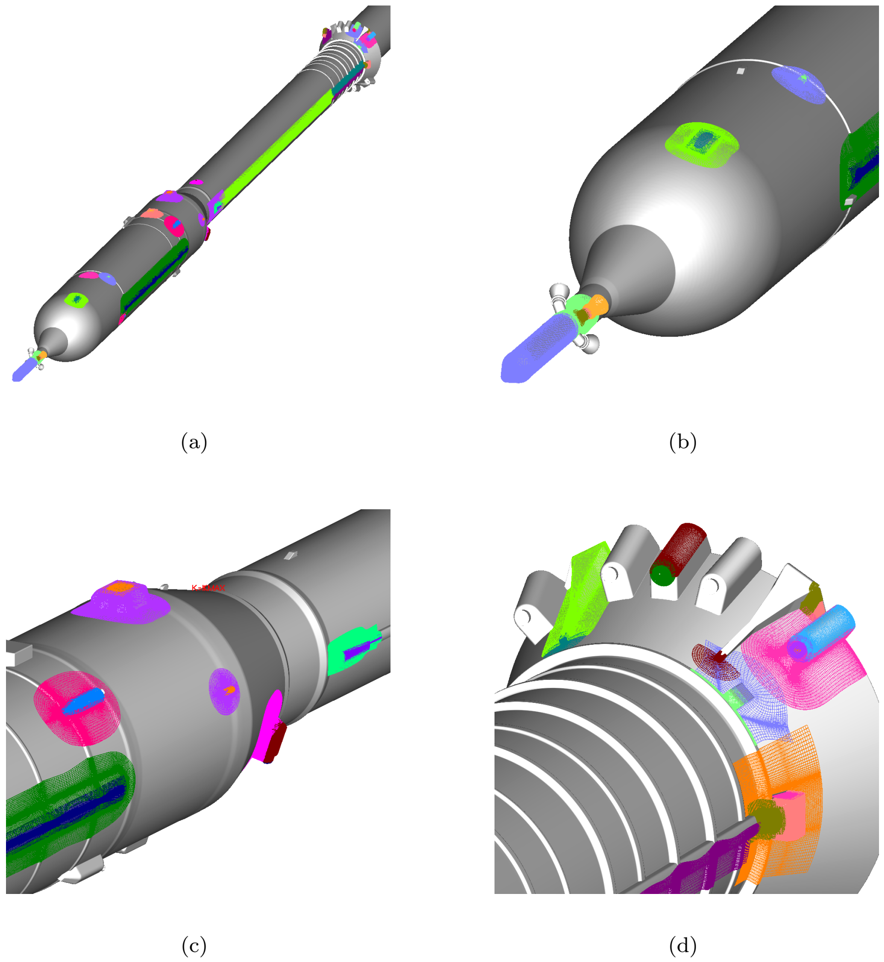
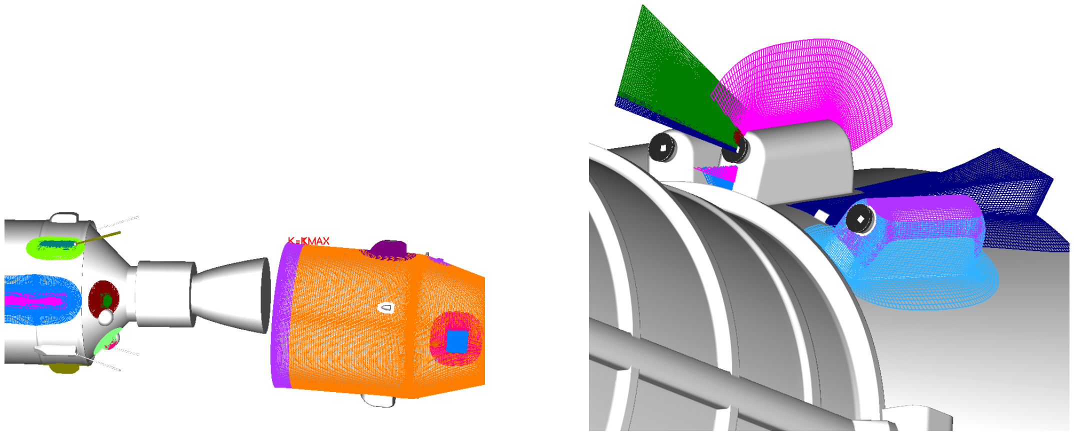


Figure 4. Surface grids for Integrated Crew Launch Vehicle and various protuberances (a) Complete stack. (b) Front region showing launch abort vehicle. (c) Upper-stage and inter-stage region. (d) First-stage separation motors region.

cavity and protuberances in the interior of the inter-stage in addition to the baseline body and protuberances (Fig. 5a). Moreover, plume grids were constructed using the *BuildGeneralPlumeGrids* macro for the upper and first stage nozzles, as well as the ullage, tumble and booster-deceleration motors (Fig. 5b). Special curvilinear off-body grids were constructed in the script to resolve the plume interaction region between the ullage and booster-deceleration motors plumes during separation. The final grid system consisted of 224 volume grids, 267 million grid points and 142 component X-rays. The resulting scripts were used to create grid systems for many relative positions of the first and upper stages in steady-state database generation, as well as for 6-dof dynamic separation computations.[25]

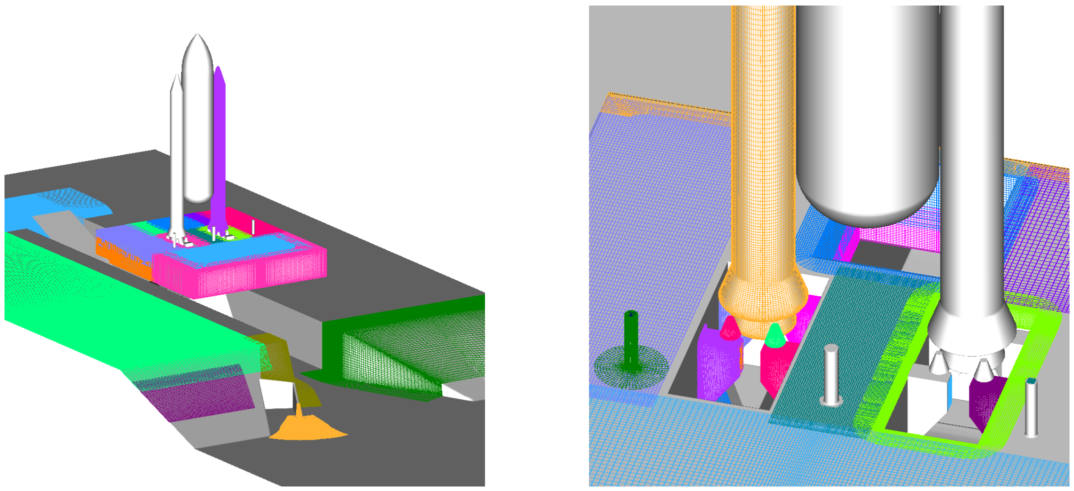
II.B.3. Launch Site Simulation

Simulations of the launch site at Kennedy Space Center were performed by the NASA Ames launch site simulation team to study ignition overpressure for various configurations of the Mobile Launch Platform (MLP) for current and possible future launch vehicles.[6] The geometry modeling included the flame trench and its immediate surrounding ground terrain, the plume deflectors, the MLP with plume cavities and support structures, and various launch vehicle configurations. Fig. 6 shows the surface grid system for a configuration containing just the External Tank and Solid Rocket Boosters from the Space Shuttle Launch Vehicle. The final grid system consisted of 129 volume grids, 92 million grid points and 32 component X-rays.



(a) (b)

Figure 5. Surface and plume grids for Crew Launch Vehicle and various protuberances in stage-separation mode (a) Upper-stage and inter-stage separation region. (b) Plume grids for first-stage separation motors.

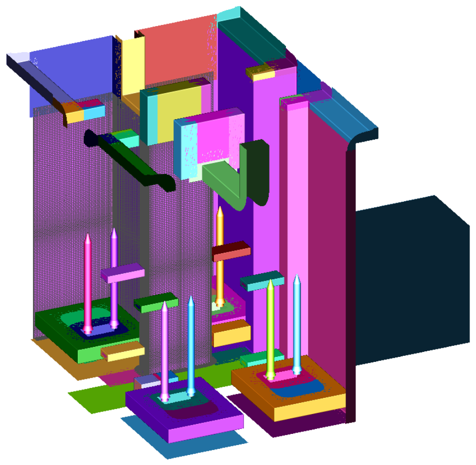


(a) (b)

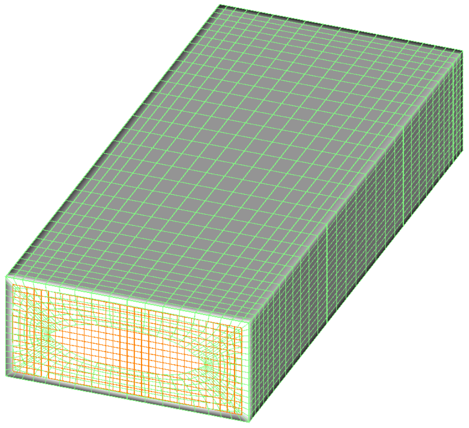
Figure 6. Surface grids for launch site flame trench ignition overpressure simulation (a) Launch site surrounding terrain, flame trench, MLP, and launch vehicle. (b) MLP cavity structures.

II.B.4. Vehicle Assembly Building Interior

In the modeling and simulation of air flow inside the Vehicle Assembly Building, a script was developed for most of the pre-processing steps. A simplified version of the interior of the building was modeled by including only the major features of the interior walls, the concrete separators between the high bays, and a Mobile Launch Platform (MLP) with Solid Rocket Boosters (SRBs) in each of the four bays (Fig. 7). The geometry creation macros of the CGT script library was used to create most of the grids except for the SRBs. The resulting grid system contained 160 overset volume grids with a total of 91 million grid points, and 49 component X-rays.



(a)



(b)

Figure 7. Surface grids for Vehicle Assembly Building interior. (a) Interior bays, concrete dividers, Mobile Launch Platforms and Solid Rocket Boosters. (b) Close-up view of one of the concrete blocks modeled as a rectangular prism.

III. Domain Connectivity Strategy and Tools

Domain connectivity or grid assembly is composed of two main steps. First, field grid points from a grid that fall inside the solid boundary of a body, or outside the computational domain are tagged or blanked. Grid points on the boundary of the resulting 'holes', and the outer boundary points of grids that do not receive flow solver boundary conditions are called fringe points (see Fig. 8). For optimal inter-grid communication, such holes need to be expanded or contracted such that the overlap between neighboring grids occur in regions where the cell sizes of neighboring grids are compatible. The second step involves the determination of interpolation stencils for the fringe points from neighboring overlapped grids.

Early efforts in domain connectivity software were heavily labor intensive requiring long manually created input files or graphical user interface work, and extensive user expertise.[26,27] More recent developments have significantly reduced the effort required from the user. These include PEGASUS5,[28] object X-rays,[7] SUGGAR++,[29] OVERTURE,[30] and PUNDIT.[31] However, as the complexity of today's applications continues to increase, the user's expertise and effort required to operate such software remain non-trivial.

In order to handle today's complex applications involving both steady and unsteady relative-body-motion problems, the ideal domain connectivity software should possess the following attributes:

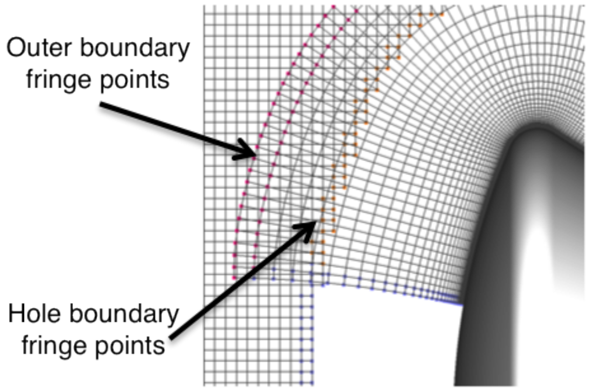


Figure 8. Slices of volume grids showing fringe points at the outer boundaries (magenta and blue symbols) and hole boundaries (orange symbols).

1. Minimal user expertise and effort are needed to create the necessary input files.
2. The software is sufficiently robust such that failure is rare.
3. The software operation is automated to a level where user intervention is not needed except for the specification of the input files.
4. The execution of the software should be fast so that it can be efficiently utilized in problems involving bodies in relative motion where domain connectivity is performed at every time step.
5. The memory usage should be sufficiently low so that very large problems could be handled with standard installed hardware memory.

6. The main functions of the software should be developed as a library that can be easily linked to a stand-alone main driver program, or to another program such as a grid generator for determining adequacy of overlap, or to a flow solver to be used in problems involving bodies in relative motion.

III.A. Chimera Components Connectivity Library

Many of the existing domain connectivity software in use today possess some of the attributes listed at the end of the previous section, but none of them satisfactorily demonstrates all of the qualities desired. The object X-rays approach[7] is robust, fast, and has been successfully utilized in moving-body problems.[5,23] However, the inputs require substantial user expertise and effort. It was recognized that many of the steps involved in the object X-rays approach could be automated, and certain details of the algorithm could be improved. The Chimera Components Connectivity Library (C3LIB) was developed in Fortran 90 with the near-term goal of creating an automated ray-tracing-based approach to hole-cutting, while trying to include all of the attributes described above.

The main strategy is to establish connectivity between components that represent various parts of the geometry in the simulation. Each component consists of surface subsets that may or may not form a closed surface, and each component is allowed to cut holes in grid subsets in the volume grid system based on simple component/grid relationship rules.[8] This library is currently developed for structured overset grids but many of the concepts and algorithms involved are easily extensible to unstructured overset grids. A longer term goal, only partially covered by this paper, is to investigate better ways to perform each step of the process, and implement the best ideas from different sources into the library.

The various domain connectivity functions in C3LIB are called from a main driver program C3P (Chimera Components Connectivity Program). The inputs required to run C3P have been designed to be as simple as possible. Two of these inputs are also inputs for the flow solver: a given set of overlapping structured volume grids, and the specification of surface subsets where flow solver boundary conditions are to be applied. The remaining two inputs are relatively easy to create: the surface subsets that make up each component, and the grid type of each grid in the volume grid system (uniform Cartesian, stretched Cartesian, or curvilinear). It is worthwhile to note that while the user still has to manually prepare the specification of component surface subsets, this step could potentially be further automated if the surface grid cells retained information on the geometric component that each grid cell resided on. This would require an additional step in the surface grid generation procedure where each grid cell retains an integer tag that stores the identification number of the geometric component from which the grid cell is derived. Such a scheme is currently not implemented but could be considered a best practice in future surface grid generation procedures.

With such inputs, domain connectivity using C3P is automated by first performing hole cutting on the given set of overlapping volume grids using an oriented or on-demand ray-shooting approach. Details on this method are given in Ref. 8. Interpolation stencil searches are then performed. Details on investigations into new ideas for stencil search methods are presented in the Sections III.B, C, and D in the current paper.

All of the common functions needed for domain connectivity are created in library form in C3LIB with a standard Applications Programming Interface (API). At the lowest level of this library are stencil search functions that are utilized by higher level domain connectivity functions. The three main stencil-search related functions are:

1. Creation of data structures needed for surface and volume grids stencil search in Fortran 90 modules.
2. Given a point (x_p, y_p, z_p) near a surface domain, determine the interpolation stencil(s) for this point in the surface domain consisting of multiple structured surface grids (see Section III.B).
3. Given a point (x_p, y_p, z_p) in a volume domain, determine the interpolation stencil(s) for this point in the volume domain consisting of multiple structured volume grids (see Section III.D).

For function (3), all possible interpolation stencils for a given fringe point are located and the best one is chosen based on specific selection criteria. As a first cut, the criterion used is that the stencil that best resolves the flow solution, i.e., the one with the smallest cell volume (finest mesh) is selected. Additionally, an interpolation stencil for a fringe point is considered valid only if the stencil quality Q is greater than or equal to a user-specfied threshold Q_{min}. The stencil quality Q is defined to be zero if one or more of the corner points in the interpolation cell is a blanked point. Otherwise, for a fringe point located inside a cell with index j, k, l at the lower corner, Q has the range $0 \leq Q \leq 1$ and is defined by

$$\begin{aligned} Q \;=\; & (1-\xi)(1-\eta)(1-\zeta) \times B_{j,k,l} + \xi(1-\eta)(1-\zeta) \times B_{j+1,k,l} + \\ & (1-\xi)\eta(1-\zeta) \times B_{j,k+1,l} + \xi\eta(1-\zeta) \times B_{j+1,k+1,l} + \\ & (1-\xi)(1-\eta)\zeta \times B_{j,k,l+1} + \xi(1-\eta)\zeta \times B_{j+1,k,l+1} + \\ & (1-\xi)\eta\zeta \times B_{j,k+1,l+1} + \xi\eta\zeta \times B_{j+1,k+1,l+1} \end{aligned} \tag{1}$$

where $0 \leq \xi, \eta, \zeta \leq 1$ are the trilinear interpolation coefficients of the fringe point in the cell, and

$$\begin{aligned} B_{j,k,l} &= 1, \quad \text{if grid point (j,k,l) is a regular field point,} \\ &= 0, \quad \text{if grid point (j,k,l) is a fringe point.} \end{aligned} \tag{2}$$

Functions (2) and (3) above are utilized by one or more of the following higher level routines:

1. Creation of the minimum holes in all grids from solid boundaries of each component by marking up an "iblanks" array. This step serves to tag grid points from any grid that fall inside the solid boundaries of each component.
2. Determination of the index locations of outer boundary and hole boundary fringe points for each grid where the number of layers of fringe points is user-prescribed.
3. Determination of interpolation stencils for all fringe points.
4. Adjustments of the existing holes to achieve 'optimized' overlap between grids. This overlap or hole optimization step is needed to prevent grid communication from occurring between grids in regions of disparate grid resolutions, e.g., coarse off-body Cartesian grid points in the fine viscous boundary layer region of near-body curvilinear grids. Such disparate grid communication, if allowed to happen, could cause degradation in solution accuracy and convergence.[32]
5. Performance of fringe point shifting near the solid walls of overlapped structured grids[28] (see Section III.C).

III.B. Surface Domain Stencil Search Procedure

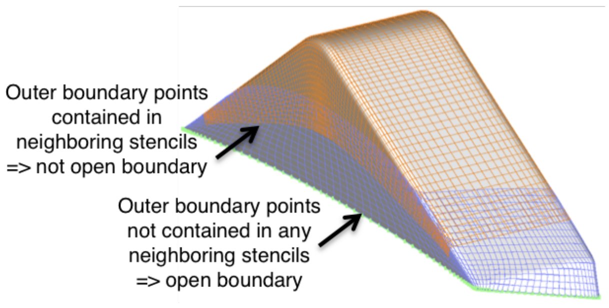
Given any point P (fringe or otherwise) in a 3-D domain near the surface of a set of overlapping surface grids, the surface stencil search function is used to locate the quadrilateral surface grid cell (interpolation stencil) that contains P using a two-level bounding box scheme similar to that described in Ref. 14. The surface stencil search function is utilized by two other domain connectivity tasks: the minimum hole cut and the solid-wall fringe-point shifting functions. The automated minimum hole cut procedure described in Ref. 8 requires the identification of vertices that lie on open boundaries of a set of surface grids that belong to a geometric component. By searching for surface interpolation stencils at all outer boundary vertices of these surface grids, any vertex on such boundaries that could not find an interpolation stencil is marked as a vertex on an open boundary (Fig. 9a). Applications of the surface stencil search function to solid-wall fringe-point shifting is described in Section III.C (Fig. 9b).

The surface stencil search function was applied to the surface grids of the entire Ares-I configuration shown in Fig. 4a. It took 9 CPU seconds on one processor of an Intel Xeon Linux workstation to locate 134000 surface stencils on 2 million surface grid points. Using OpenMP on 8 processors, the same problem took 1.1 seconds of wall-clock time to complete.

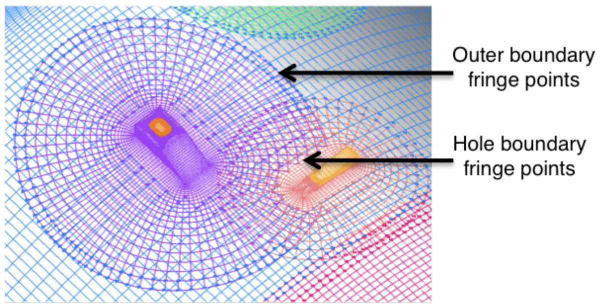
III.C. Fringe Point Shifting Near Solid Wall Boundaries

Prior to seeking interpolation stencils for fringe points in the volume grid system, special treatment is needed for the fringe points near solid wall boundaries. This is because slight mismatches in the discrete representation of same solid wall boundary by adjacent overlapping grids result in the solid wall surface grid points of one grid sitting slightly above or below surface grid cells of a neighboring overlapping grid. As a result, some of these fringe points will receive no interpolation stencils. The solution to this problem is discussed in Ref. 28, and a brief outline of the procedure is given below.

First, the outer and hole boundary fringe points of overlapping grids at the wall boundaries are projected onto surface donor stencils of neighboring grids. This projection is performed using the surface stencil search



(a)



(b)

Figure 9. Applications of surface stencil search function. (a) Identification of points on open boundaries of a group of surface grids that make up a component. (b) Identification of surface fringe point interpolation stencils for fringe point shifting near solid wall boundaries.

function described in Section III.B. Fringe points along a grid line above a surface fringe point are shifted by the same vector but with a quadratic decay factor with index distance from the wall. These shifted fringe points are combined with the other fringe points in the volume grid and interpolation stencils are sought using the procedure describe in the next section. While the interpolation stencils are determined using the coordinates of the shifted fringe points, the original coordinates of these fringe points are left untouched in the grid system to maintain the surface geometry discretization that is initially specified.

A parallel procedure is used to determine the surface fringe point interpolation stencils using OpenMP by setting the parallel loop over all surface fringe points. For all the test cases described in the rest of this paper, the fringe point shifting step is cheap and costs only 1% or less of the total volume fringe points stencil search time.

III.D. Volume Domain Stencil Search Procedure

Given any point P (fringe or otherwise) in a 3-D domain of overlapping volume grids, the volume stencil search function is used to locate the hexahedral grid cell (interpolation stencil) that contains P. In some cases, multiple valid interpolation stencils may be found. It is important to select an interpolation stencil of compatible cell size to the fringe point in such situations.[28] For example, a fringe point from a fine grid A may find a stencil from a coarse grid B, and from another fine grid C with similar grid resolution as A. If the coarse grid stencil is selected, high flow gradients in the fine grid A that cannot be supported in the coarse grid will be "blocked" by the coarse grid interpolation. On the other hand, if the stencil from the fine grid C is chosen, then the high flow gradient from A would be able to pass on to C. For the current work, the stencil with the smallest cell size is selected to avoid the coarse grid "blocking" problem. A "cell-difference parameter" function was used in Ref. 28 for determining the best stencil to accept. While this scheme has worked well for many situations, further research may be needed to determine if this is always the most appropriate function to employ.

The volume stencil search function is utilized in the search for interpolation stencils for all fringe points in all grids. Various stencil search algorithms have been adopted by different software in use today. Typically, the algorithm uses a global filter such as bounding boxes and/or various ADT/octree data structures to quickly narrow down the search to a local region. Then, a local stencil walk method based on gradient search is used to locate the interpolation stencil that contains the target point.[33] The trilinear interpolation coefficients for the cells encountered in the stencil walk are used to direct the gradient search.

The current work involves the investigation of new ideas to further speed up the stencil search algorithm. It is recognized that most search algorithms require a relatively expensive global filter followed by a cheaper local stencil walk for every fringe point. Clearly, the cost of the stencil search can be reduced if a more efficient global filter is utilized. Furthermore, if it is possible to eliminate the global filter step for as many fringe points as possible, then the total process time will be reduced. The proposed algorithm outlined below tries to accomplish this by using sub-level bounding boxes (sub-boxes) and constructing space-filling path segments that connect as many neighboring fringe points as possible.

III.D.1. Search Algorithm

An outline of the search algorithm is given below which will determine all possible interpolation stencils for each fringe point. As a pre-processing step, for each grid G in the given volume grid system, a list of sub-boxes is created by an approximately equal partition of the grid index space in the j, k, and l directions (see Fig.10). This results in an approximately equal number of potential interpolation donor cells in each sub-box, and hence an approximately balanced search procedure.

Then for each grid G, the following is performed.

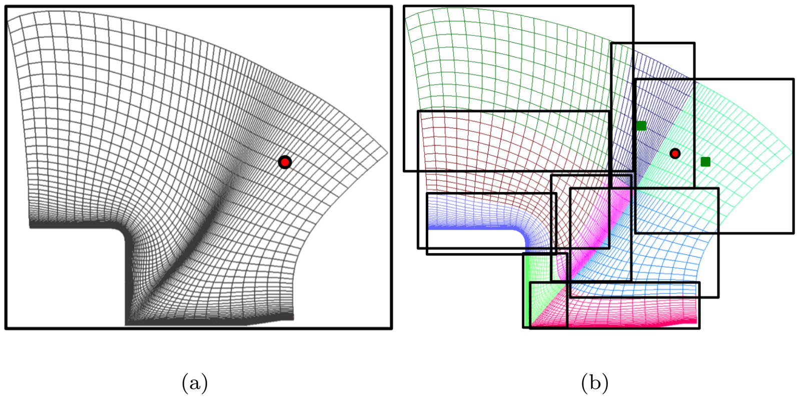


Figure 10. Two-dimensional example showing global filter with sub-boxes based on balanced index-space partitions. (a) Curvilinear grid cells and grid bounding box. Fringe point denoted by red symbol. (b) Sub-boxes based on balanced index-space partitions. Green symbols denote starting points of local stencil walk from the two sub-boxes that contain the red fringe point.

1. For each fringe point P in G, use the global bounding boxes of each neighboring grid to build a list of all grids that may contain an interpolation stencil for P. Let the number of grid bounding boxes that contains P be M_{st}, and the actual number of grids with interpolation stencils that contain P be N_{st}. Then, we know $N_{st} \leq M_{st}$.
2. By combining the grid lists created in step 1 for all fringe points in G, generate a master list (L) of all grids that may supply interpolation stencils to all fringe points in G.
3. For each grid N in L,
   (a) Identify all fringe points in G that are contained by the bounding box of grid N.
   (b) If grid N is a uniform Cartesian grid, the interpolation stencil is immediately located by a direct evaluation method. The interpolation cell lower corner indices j, k, l are given by

$$j = int[\frac{(x_p - x_0)}{\Delta x}] + 1, \quad k = int[\frac{(y_p - y_0)}{\Delta y}] + 1, \quad l = int[\frac{(z_p - z_0)}{\Delta z}] + 1, \tag{3}$$

   where int is a truncation function to an integer towards zero, x_0, y_0, z_0 are the coordinates of the lowest corner of the Cartesian grid bounding box, and $\Delta x, \Delta y, \Delta z$ are the uniform grid spacings in the x, y, and z directions, respectively.
   (c) If grid N is a stretched Cartesian grid, the interpolation stencil cell indices j, k, l are immediately located by one-dimensional searches in each of the x, y, and z directions, respectively.
   (d) If grid N is non-Cartesian (curvilinear), create a space-filling path segment by finding an unused fringe point P_0 as the first point of the segment. Build the path from P_0 by walking to another fringe point within one cell away. Continue walking and connecting fringe points until there are no fringe points within one cell away, then terminate the path segment. Start another path segment by returning to step 3(d) until all fringe points are used. For each path segment, follow step 3(e) to find interpolation stencils for the fringe points on the path.
   (e) The interpolation stencil for the first point P_0 of a path segment is found using a global filter by locating all sub-boxes of grid N that contain the point. The center point in index space of a sub-box that contains P_0 is used to start a local stencil walk to locate the interpolation stencil for P_0. Note that this avoids having to perform an expensive nearest point test to begin the stencil walk. If the stencil walk fails to converge, then the nearest point inside the sub-box is used to begin another local stencil walk. If this second stencil walk still fails to converge, then the search jumps to another sub-box that contains P_0 until all such sub-boxes have been considered. For all points on the path segment subsequent to P_0, use the previous point's interpolation stencil as the starting guess for a local stencil walk to find the stencil for the current point.

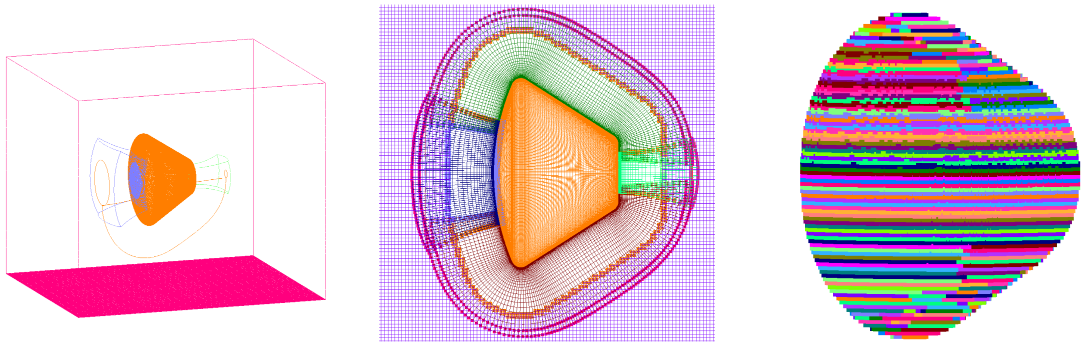


Figure 11. Capsule grid system (a) Volume grids: three curvilinear near-body and one Cartesian off-body. (b) Cut through symmetry plane showing fringe points (symbols). (c) Layers of fringe points in off-body Cartesian grid.

(f) If at any point on a path segment where an interpolation stencil cannot be found, the search for this fringe point's interpolation stencil is stopped here. Its stencil may be found under another grid. The processing now skips to the next point on the path segment. A global filter is applied to begin the stencil search for this point, followed by a local stencil walk. The subsequent point on the path now can go back to using a local stencil walk only, and the procedure continues until all paths are completed. Parallelization of the stencil search procedure is done using an OpenMP loop over the path segments where each processor is responsible for stencil searching for one of the path segments on the list of paths.

Fig. 11a shows a four-grid capsule example with three near-body curvilinear capsule grids embedded inside a Cartesian grid. Fringe points for the four grids are shown in a slice through the symmetry plane in Fig. 11b. The hole boundary fringe points for the Cartesian off-body grid are shown in layers in Fig. 11c.

One of the lower layers of fringe points in Fig. 11c is shown in more detail in Fig. 12. The unconnected fringe points are shown in Fig. 12a. By applying the above algorithm, the fringe points are connected by three path segments shown in Fig. 12b. A global filter is used for the first point in each of the path segments, followed by a local stencil walk to identify its interpolation stencil in grid N.

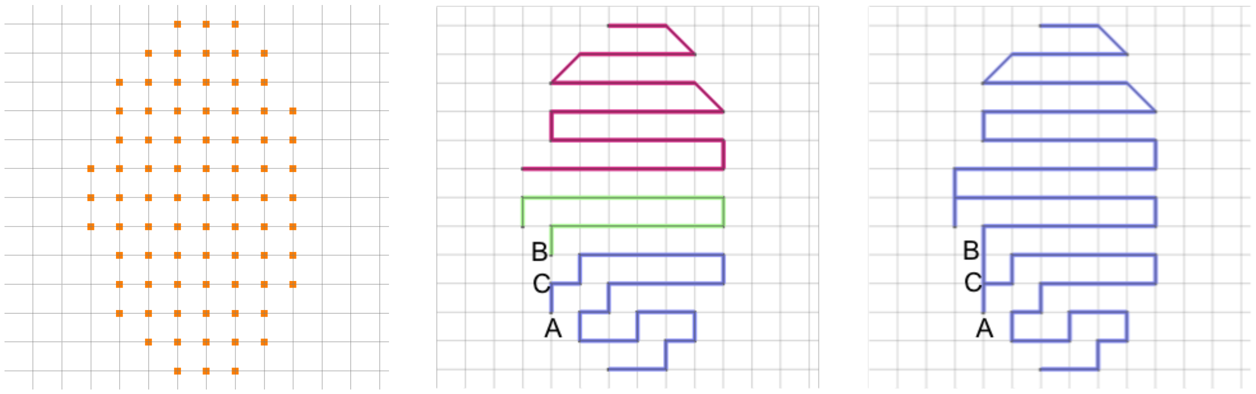


Figure 12. A layer of fringe points and connecting path segments from lower part of capsule off-body Cartesian grid. (a) Unconnected fringe points. (b) Auto-connected three path segments through fringe points when no jumps are allowed. End of blue segment (point A) is not allowed to jump to the beginning of the next segment (point B). (c) Auto-connected single path segment through fringe points when jumps are allowed. Point A is allowed to jump to point B since point B is adjacent to a previously visited point on the path (point C).

It was recognized that one could avoid global filters further by allowing jumps in the space-filling paths. Instead of a path segment having to terminate due to the absence of a fringe point within one cell away from the end of the path (point A in Fig. 12b), the path segment could be allowed to jump to another unused fringe point B elsewhere as long as point B is adjacent to another fringe point C that has already been visited by the space-filling path (Figs. 12b, c). Then the stencil walk continues from point B using the interpolation stencil from point C as the starting guess. Multiple jumps are allowed in the same path segment until the path segment has to terminate and no valid candidate points are found for further jumps. This jump option then results in longer path segments and less global filter searches. Another example comparing path segments without and with jumps is shown in Figs. 13a, b.

Table 1 shows the wall clock time on an Intel Xeon Linux workstation for interpolation stencil search for the capsule grid system. Since all possible interpolation stencils are located for each fringe point, the sum of the number of Cartesian and curvilinear stencil searches is greater than the number of fringe points. The wall clock time reported is essentially the time required for the curvilinear stencil searches since the Cartesian stencil searches are about a factor of 100 faster. It can be seen that significant time savings is achieved by activating the space-filling path segments option as opposed to performing the global filter and local walk at every fringe point. In this case, activating the jump option versus no jumps does not appear to make much difference since this is a relatively small problem and the code is already operating very fast.

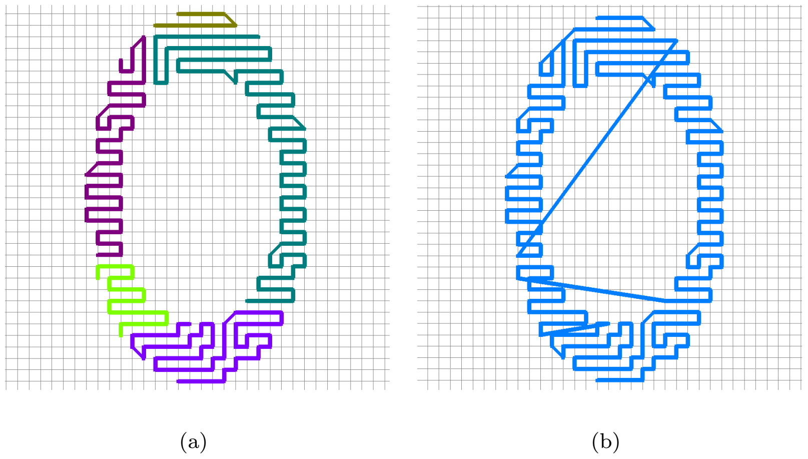


Figure 13. A layer of fringe points and connecting path segments from middle part of capsule off-body Cartesain grid. (a) Auto-connected path segments through fringe points when no jumps are allowed. (b) Auto-connected single path segment through fringe points when jumps are allowed.

Table 1. Wall clock time in seconds on Intel Xeon workstation for stencil search procedure with the default OpenMP loop balancing schedule for capsule grid system (4 grids, 6 million volume grid points, 270000 fringe points, 240000 Cartesian stencil searches, 200000 curvilinear stencil searches).

Number of processors	**Global filter + local walk**	**Space-filling path (no jumps)**	**Space-filling path (with jumps)**
1	1.30	0.74	0.74
8	1.06	0.49	0.51

Table 2. Wall clock time in seconds on Intel Xeon workstation for stencil search procedure with various OpenMP loop balancing schedules for front part of crew launch vehicle grid system (28 grids, 31 million volume grid points, 2 million fringe points, 1.9 million Cartesian stencil searches, 2.9 million curvilinear stencil searches).

Number of processors (loop schedule)	**Global search for all points**	**Space-filling path, no jumps**	**Space-filling path, with jumps**
1 (default)	56.5	46.4	45.0
4 (default)	24.4	24.9	24.3
8 (default)	18.0	17.8	18.3
8 (static,1)	17.1	14.8	14.9
8 (dynamic,1)	17.7	14.7	16.0

Table 2 shows the wall clock time for interpolation stencil search for the front section of the Ares-I grid system (Fig. 4b) which is a substantially larger system than the capsule. Similar to the capsule case, the space-filling path options result in faster search times than the global search option using 1 processor, or 8 processors with the OpenMP loop schedule specified to be static or dynamic with chunk size one. Again, no significant gains are observed by allowing jumps in the space-filling path option. However, the default loop schedule for OpenMP with the Portland Group compiler appears to be not optimal. By switching to the static or dynamic schedule with chunk size one, faster search time are achieved compared to the default schedule. This is expected since the work load for the different path segments (and hence that for each processor) can be quite different, and a dynamic schedule should provide a faster run time.

The third test case performed is a high-lift system consisting of a fuselage, wing, slat, and flap with 28 near-body volume grids, 298 off-body Cartesian grids, 36 million grid points and 8.2 million fringe points (Figs. 14a,b) Using 8 processors, the jump option took 196 seconds to complete the stencil search, while the no-jump option took only 168 seconds. This result contradicts the expectation that the jump option should be faster with less global searches than the no-jump option. Further analysis is shown in the next sub-section to explain this behavior.

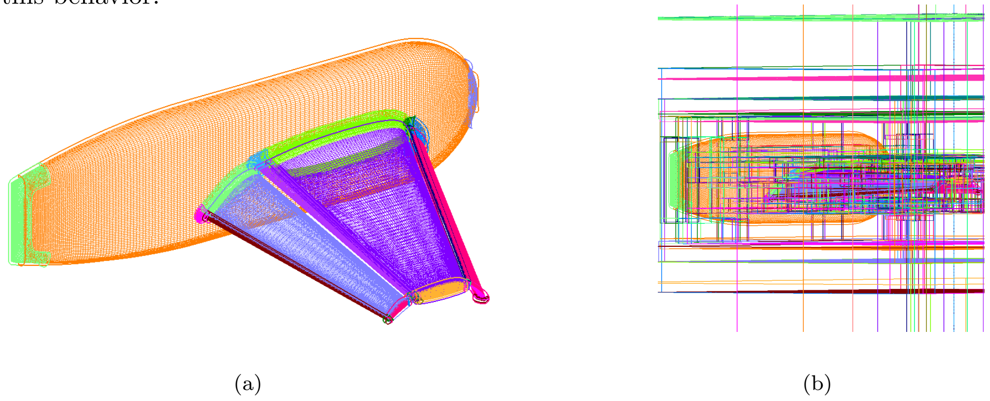

(a) (b)

Figure 14. High-lift grid system. (a) Near-body grids. (b) Off-body grids with successive layers of refinement.

III.D.2. Analysis of Differences in Space-Filling Path Jump Options

In the test cases presented so far, activating the jump option in the space-filling path stencil search either shows no speed-up or a degradation in speed. This behavior can be explained by looking more carefully at the expected gains in time on reducing the total number of global searches under the jump option. First, the cost of a global search per fringe point is determined to be about $17 \times 10^{-6}s$ from a test case using a single processor with the option where all stencils are computed using global searches. Running the same case with the space-filling path option on one processor requires a combination of global and local searches. From the wall clock time needed to complete this case, the cost of a local search is determined to be about $11 \times 10^{-6}s$. The difference in time between a global and local search is then $6 \times 10^{-6}s$. In other words, a reduction in wall clock time of 1 second requires converting about 167000 global searches to local searches.

For the high-lift test case presented in the previous section, switching from the no-jump option to the jump option converted only 14500 global searches to local searches. This suggests that we should see a difference in wall clock time of only about a tenth of a second. With 8 processors running OpenMP, a difference of 28 seconds was observed. Some other mechanism must be at work to cause this discrepancy. A deeper analysis on the segmented space-filling paths may reveal an explanation.

Recall from Section III.D.1 that the fringe points for each grid in the volume grid system are grouped by the bounding boxes of the neighboring grids that they fall in. For each pairing of a grid G and one of its neighboring grids whose bounding box contains at least one fringe point from G, a space-filling search path is constructed. This grid pairing is denoted a "search packet". For each search packet, the space-filling path is divided into segments with either the "jump""or the "no-jump" option. Fig. 15a shows the mean of the difference in the space-filling path segment lengths between the "jump" and the "no-jump" option. With the mean of the difference being always positive, this implies that the "jump" option segment lengths are always longer than those in the "no-jump" option as expected. Fig. 15b shows the standard deviation of the difference in the segment lengths between the "jump" and "no-jump" option. With the standard deviation of the difference being positive in most cases, this implies that the "jump" option has a much larger range of segment lengths compared to the "no-jump" option. This larger range of segment lengths is causing imbalances in the OpenMP loops that apparently cannot be compensated efficiently by the dynamic loop schedule. The result is a noticeably slower run time for the "jump" option.

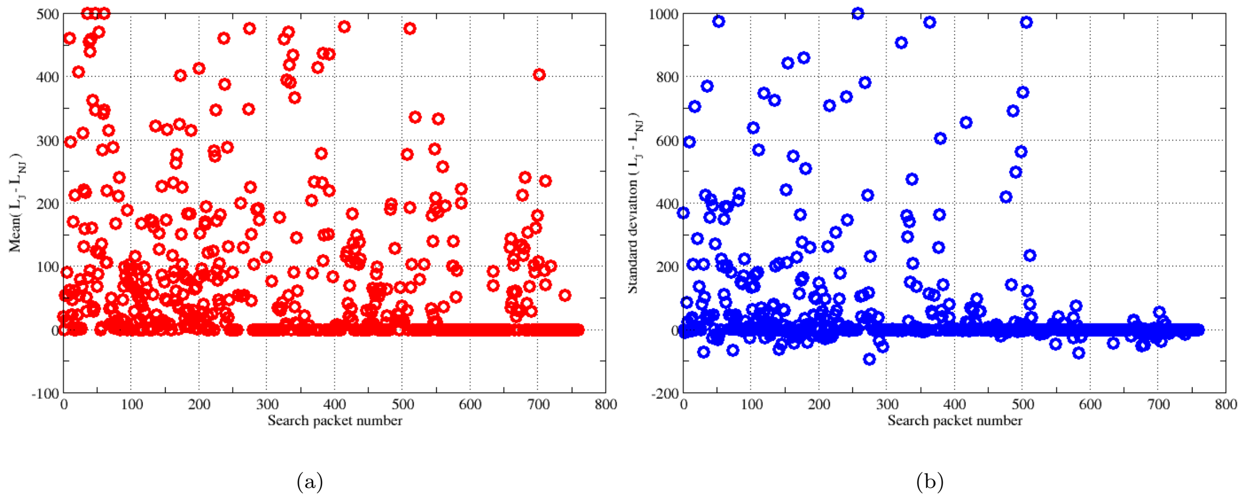


Figure 15. Space-filling path statistics for high-lift test case for jump and no-jump options. L_J and L_{NJ} are the lengths of the space-filling path segments for the jump and no-jump options, respectively. (a) Mean of differences in segment lengths between jump and no-jump option. (b) Standard deviation of differences in segment lengths between jump and no-jump option.

III.E. Comparison with Original X-rays Stencil Search Procedure

The original X-ray hole-cutting and stencil search procedure implemented in the OVERFLOW flow solver[13] accepts the first interpolation stencil that it can find for each fringe point and looks no further. No attempt is made to locate the best stencil based on cell compatibility if multiple stencils are available. The current C3LIB procedure follows a practice similar to the PEGASUS5[28] code and searches for all possible stencils and accepts the one with the smallest cell size. It is expected that this scheme, although more robust, will be more expensive than the one in OVERFLOW. A comparison of the timings and memory usages for several large test cases are presented below using the no-jump option in C3LIB.

The first is the high-lift test case shown in Section III.D.1. Table 3a shows that C3LIB takes about 50% longer to find the stencils compared to OVERFLOW but the memory usage is about 25% less. The second test case is the Ares-I grid system with full protuberances (Fig. 4). There are 190 volume grids, 153 million grid points, and 12.3 million fringe points. Table 3b shows that C3LIB takes about 20% longer to find the stencils compared to OVERFLOW while the memory usage is about 40% less. The third test case is a Saturn-V grid system (Fig. 16) with 83 volume grids, 286 million grid points, and 13.5 million fringe points. Table 3c shows that C3LIB is about 3.5 times faster than OVERFLOW in stencil searches, and the memory usage is about 30% less.

III.F. Comparison of Different Compilers

For all the test cases presented so far, the timing results were obtained using the Portland Group Fortran 90 compiler (*pgf90* version 10.6). Further speed up of C3LIB has been accomplished using the GNU Fortran compiler (*gfortran* version 4.1), and the Intel Fortran compiler (*ifort* version 11.1). For all three compilers, the standard "-O" compiler flag was used, and no attempt was made to investigate more aggressive optimization flags. Identical interpolation stencils were obtained for the three compilers. Table 4 summarizes the performance of these three compilers on five test cases presented in earlier sections. The results are given for interpolation stencil search time only, and for the total execution time which includes I/O, fringe point identification, fringe point shifting near solid walls, and various other overheads.

For the test cases performed, *ifort* produced the fastest execution time. It is about 15% to 37% faster than *pgf90* in both stencil search and total time. Also, compared to *gfortran*, it is about 3% to 16% faster for stencil search, and 13% to 69% faster for total time. For stencil search only, *gfortran* seems to perform consistently better than *pgf90*, and is almost as good as *ifort* for most cases. For the total time, *gfortran*'s performance appears to be more erratic. It is slower than *pgf90* for the two smaller cases and slightly faster than *pgf90* for the larger three cases. Further code profiling seems to suggest slower I/O performance as one of main reasons for *gfortran*'s behavior for the total time.

Table 3. Comparison of stencil search times and memory usage between the original scheme in OVERFLOW and current scheme (no-jump) in C3LIB for three test cases. Wall clock time is in seconds on an Intel Xeon workstation using 8 processors.

Code (parallel scheme)	Stencil Search	Wall Time (s)	Memory Usage (GB)
OVERFLOW/DCF (MPI)	find first	109	4.7
C3LIB (OpenMP)	find all	169	3.7

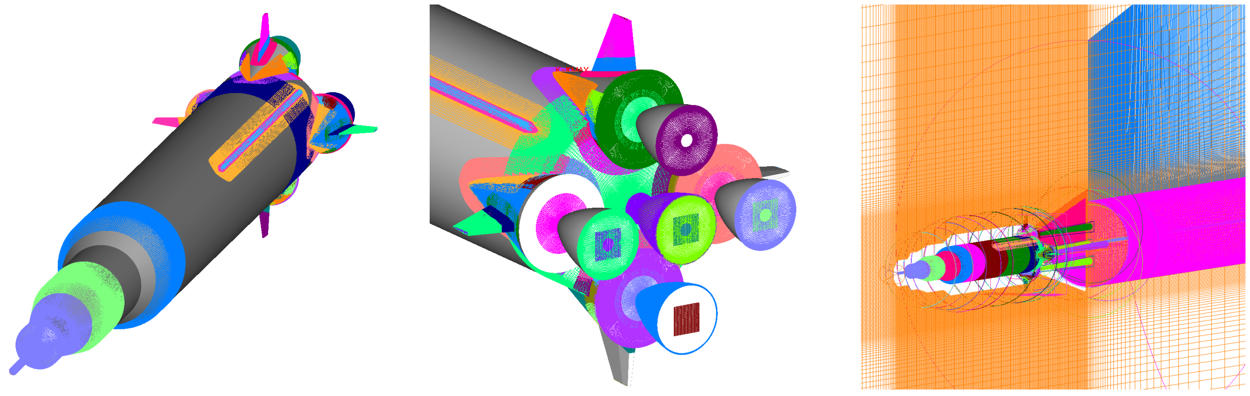
(a) High-lift

Code (parallel scheme)	Stencil Search	Wall Time (s)	Memory Usage (GB)
OVERFLOW/DCF (MPI)	find first	249	16.5
C3LIB (OpenMP)	find all	295	10.0

(b) Ares-I

Code (parallel scheme)	Stencil Search	Wall Time (s)	Memory Usage (GB)
OVERFLOW/DCF (MPI)	find first	876	26.3
C3LIB (OpenMP)	find all	251	18.1

(c) Saturn-V



(a) (b) (c)

Figure 16. Saturn-V grid system. (a) Surface grids (front view). (b) Surface grids (back view). (c) Near and off-body volume grids.

Table 4. Wall clock time in seconds on Intel Xeon workstation using 8 processors with OpenMP for stencil search on five test cases using three different compilers.

	Stencil Search Only (sec.)			Total (sec.)		
Test case	pgf90	gfortran	ifort	pgf90	gfortran	ifort
Capsule	0.49	0.44	0.37	1.6	3.5	1.1
Crew launch vehicle (front)	13.1	9.3	8.5	19.7	24.1	12.4
High lift system	169	149	144	181	171	149
Crew launch vehicle (full)	295	241	226	350	325	246
Saturn-V	251	184	174	359	313	251

IV. Summary and Conclusions

The current strategy adopted for reducing the grid generation time for overset structured grid generation on complex configurations is described. A scripting approach based on a sophisticated hierarchical library of low and high level grid generation macros is used to speed up the process. While it is not a general automated approach, the scripting method does offer a convenient and rapid means to re-generate a grid system with parameterized inputs. The main draw-back is that the script needs to be manually created the first time and requires modification if a topological change occurs in the geometry. With the development of a more comprehensive script library and higher level macros, the scripting approach has been successfully demonstrated on a variety of complex geometries.

A library of domain connectivity functions with standardized interfaces is proposed that strives to satisfy the criteria of robustness, automation, speed, low memory usage and modularity. The hole-cutting algorithm is described in a separate paper[8] while enhancements made to the basic stencil search algorithms for surface and volume domains are presented in the current paper. Sub-level bounding boxes were employed to improve the global filter efficiency. A space-filling path method is used to significantly reduce the number of global searches for the fringe points. These, together with OpenMP directives in a parallelized code, resulted in reduced execution time. The new library also offers more generalized functions to return all possible stencils for the fringe points, as well as a more general search procedure for fringe point shifting at the solid wall boundaries. Both of these features are enhanced stencil treatments that remove the limitations in the original object X-rays method.

In a comparison study between the stencil search scheme implemented in OVERFLOW and in C3LIB, the C3LIB scheme described in this paper falls between about 1.5 times slower to 3.5 times faster. This indicates that the C3LIB scheme is at least competitive with current methods even though it is performing a more extensive stencil search. Furthermore, the C3LIB procedure requires less memory usage (about 25-40% less) than that in OVERFLOW in all cases tested. Future work will include comparisons with similar software such as PEGASUS5 and SUGGAR. If the stencil search schemes from these codes (or from any other new ideas) turn out to be more efficient, it should then be not too difficult to implement them into the C3LIB library framework.

V. Acknowledgements

This work has been performed under NASA's Constellation Program to support aerodynamics analysis, and ground and launch operations for the next generation space exploration vehicles including both Ares-I and V. The author would like to thank Dr. Shishir Pandya for insightful discussions on the stencil search algorithm presented in this paper.

References

[1]Gomez, R. J., Vicker, D., Rogers, S. E., Aftosmis, M. J., Chan, W. M., Meakin, R. L. and Murman, S., "STS-107 Investigation Ascent CFD Support," AIAA Paper 2004–2226, 2004.

[2]Ahmad, J. U., Pandya, S. A., Chan, W. M. and Chaderjian, N. M., "Navier-Stokes Simulation of Air-Conditioning Facility of a Large Modern Computer Room," FEDSM 2005-77225, Proceedings of the 2005 ASME Fluids Engineering Division Summer Meeting and Exhibition, Houston, Texas, 2005.

[3]Pandya, S., Onufer, J., Chan, W. and Klopfer, G., "Capsule Abort Recontact Simulation," AIAA Paper 2006–3324, 2006.

[4]Kiris, C. C., Kwak, D., Chan, W. M., Housman, J. A., "High-Fidelity Simulations of Unsteady Flow Through Turbopumps and Flowliners," *Computers & Fluids*, Vol. 37, pp. 536-546, 2008.

[5]Bhagwat, M., Dimanlig, A., Saberi H., Meadowcroft, E., Panda, B. and Strawn, R., "CFD/CSD Coupled Trim Solution of the Dual-Rotor CH-47 Helicopter Including Fuselage Modeling," Proceedings of the American Helicopters Society Aeromechanics Specialist's Conference, San Francisco, 2008.

[6]Kiris, C., Housman, J., Gusman, M., Chan, W. and Kwak, D. , "Time-Accurate Computational Analysis of Ignition Overpressure in the Flame Trench," *Computational Fluid Dynamics Review*, Eds. Hafez, Oshima, Kwak, Publisher: World Scientific, 2010.

[7]Meakin, R. L., "Object X-rays for Cutting Holes in Composite Overset Structured Grids," AIAA Paper 2001–2537, 2001.

[8]Kim, N. and Chan, W. M., "Automation of Hole-Cutting for Overset Grids Using the X-rays Approach," to be presented in 20th AIAA Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 27-30, 2011.

[9]Chan, W. M., Gomez, R. J., Rogers, S. E., and Buning, P. G., "Best Practices in Overset Grid Generation," AIAA Paper 2002–3191, 2002.

[10]Chan, W. M. and Gomez, R. J., "Advances in Automatic Overset Grid Generation Around Surface Discontinuities," AIAA Paper 1999–3303, 1999.

[11]Haimes, R. and Dannenhoffer, J. F., "Control of Boundary Representation Topology in Multidisciplinary Analysis and Design," AIAA Paper 2010–1504, 2010.

[12]Dannenhoffer, J. and Haimes, R., "Automated Creation of 3-D Overset Grids Directly from Solid Models," to be presented in 20th AIAA Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 27-30, 2011.

[13]Nichols, R. H., Tramel, R. W. and Buning, P. G., "Solver and Turbulence Model Upgrades to OVERFLOW 2 for Unsteady and High-Speed Applications," AIAA Paper 2006–2824, 2006.

[14]Chan, W. M., "Enhancements to the Hybrid Mesh Approach to Surface Loads Integration on Overset Structured Grids," AIAA Paper 2009–3990, 2009.

[15]Pandya, S. A. and Chan, W. M., "Computation of Sectional Loads from Surface Triangulation and Flow Data," to be presented in 20th AIAA Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 27-30, 2011.

[16]Kao, D. L. and Chan, W. M., "OVERSMART - A Solution Monitoring and Reporting Tool for the OVERFLOW Flow Solver," AIAA Paper 2009–3998, 2009.

[17]Chan, W. M., "The OVERGRID Interface for Computational Simulations on Overset Grids," AIAA Paper 2002–3188, 2002.

[18]Pandya, S. A. and Chan, W. M., "Automation of Structured Overset Mesh Generation for Rocket Geometries," AIAA Paper 2009–3993, 2009.

[19]Welch, B. B., Jones, K. and Hobbs, J., "*Practical Programming in Tcl and Tk*," 4th Edition, Prentice Hall PTR, 2003.

[20]Abbott, I. H. and von Doenhoff, A. E., "*Theory of Wing Sections*," Dover Publications, 1959.

[21]Parks, S. J., Buning, P. G., Steger, J. L. and Chan, W. M., "Collar Grids for Intersecting Geometric Components Within the Chimera Overlapped Grid Scheme," AIAA Paper 1991–1587, 1991.

[22]Chaderjian, N. M. and Olsen, M. E., "Grid Resolution and Turbulence Model Effects on Space Capsule Navier-Stokes Simulations," AIAA Paper 2007–4562, 2007.

[23]Ahmad, J. and Chaderjian, N., "High-Order Accurate CFD/CSD Simulation of the UH60 Rotor in Forward Flight," to be presented in 29th AIAA Applied Aerodynamics Conference, Honolulu, Hawaii, June 27-30, 2011.

[24]Haimes, R. and Aftosmis, M., "On Generating High-Quality Water-Tight Triangulations Directly from CAD," Proceedings of the 8th International Conference on Numerical Grid Generation in Computational Field Simulations, Honolulu, Hawaii, 2002.

[25]Klopfer, G. H., Kless, J. E., Lee, H. C., Onufer, J. T., Pandya, S., and Chan, W. M., "Validation of Overflow for Computing Plume Effects during the Ares I Stage Separation Process," AIAA Paper 2011–170, 2011.

[26]Meakin, R. L., "A New Method for Establishing Intergrid Communication Among Systems of Overset Grids," AIAA Paper 1991–1586, 1991.

[27]Suhs, N. E. and Tramel, R. W., "PEGSUS 4.0 Users Manual," Arnold Engineering Development Center, AEDC-TR-91-8, Arnold AFB, TN, Nov. 1991.

[28]Rogers, S. E., Suhs, N. E. and Dietz, W. E., "PEGASUS5 : An Automated Pre-Processor for Overset-Grid CFD," *AIAA J.*, Vol. 41, No. 6, pp. 1037-1045, Dec. 2003.

[29]Noack, R. W., "Suggar++: An Improved General Overset Grid Assembly Capability," AIAA Paper 2009–3992, 2009.

[30]Henshaw, W. D., "Ogen: An Overlapping Grid Generator for Overture," Research Report UCRL-MA-132237, Lawrence Livermore National Laboratory, 1998.

[31]Sitaraman, J., Floros, M., Wissink, A. and Potsdam, M., "Parallel Domain Connectivity Algorithm for Unsteady Flow Computations Using Overlapping and Adaptive Grids," *J. Comp. Phys.*, Vol. 229, pp. 4703-4723, March 2010.

[32]Rogers, S. E., "Progress in High-Lift Aerodynamic Calculations," *J. of Aircraft*, Vol. 31, No. 6, pp. 1244-1251, 1994.

[33]Meakin, R. L., "Composite Overset Structured Grids," Ch. 11, *Handbook of Grid Generation*, Eds. Thompson, Soni, Weatherill, CRC Press, 1999.